NATCracker: NAT Combinations Matter

Roberto Roverso[1,2], Sameh El-Ansary[1,3], and Seif Haridi[2]
[1] Peerialism Inc., Sweden,
[2] KTH-Royal Institute of Technology, Sweden,
[3] Nile University, Egypt
{roberto,sameh}@peerialism.com, seif@it.kth.se

***Abstract*—In this paper, we report our experience in working with Network Address Translators (NATs). Traditionally, there were only 4 types of NATs. For each type, the (im)possibility of traversal is well-known. Recently, the NAT community has provided a deeper dissection of NAT behaviors resulting into at least 27 types and documented the (im)possibility of traversal for some types. There are, however, two fundamental issues that were not previously tackled by the community. First, given the more elaborate set of behaviors, it is incorrect to reason about traversing a single NAT, instead combinations must be considered and we have not found any study that comprehensively states, for every possible combination, whether direct connectivity with no relay is feasible. Such a statement is the first outcome of the paper. Second, there is a serious need for some kind of formalism to reason about NATs which is a second outcome of this paper. The results were obtained using our own scheme which is an augmentation of currently-known traversal methods. The scheme is validated by reasoning using our formalism, simulation and implementation in a real P2P network.**

I. Introduction

Dealing with Network Address Translators (NATs) is nowadays an essential need for any P2P application. The techniques used to deal with NAT have been more or less "coined" and there are several widely-used methods[1][2]. Some of them are rather a defacto standard like STUN [3],TURN [4],ICE [5].

In the context of our a P2P live video streaming application PeerTV, we are mainly concerned with media streaming using UDP and therefore the scope of this paper is UDP NAT traversal. Moreover, we are strictly interested in solutions that do not use relay, such as TURN for instance, due to the high bandwidth requirements of video streaming.

We have found lots of of previous work on the subject that aims to answer the following question: *For every t in the set of NAT types $\mathcal{T}$, which s in the set of traversal strategies $\mathcal{S}$ should be used to traverse t? The answer is of the form $f : \mathcal{T} \rightarrow \mathcal{S}$.* i.e. the following is an example with a couple of types f : { Simple Hole Punching, Port-Prediction } $\rightarrow$ { Full-Cone, Symmetric} [6].

However, the point which we found not gaining enough attention is that the presence of a feasible traversal technique that enables two peers behind NAT to communicate depends on the "combination" of the NAT types and not on the type of each peer separately. Thus, the question should be: *"Given 2 peers p_a and p_b with respective NAT types $t(p_a)$ and $t(p_b)$, which traversal strategy s is needed for p_1 and p_2 to talk? The answer is of the form $f : \mathcal{T} \times \mathcal{T} \rightarrow \mathcal{S}$"*, i.e we need to analyze traversable combinations rather than traversable types.

Most works contain a few examples of combinations for explanation purposes [6][7]. However, we have failed to find any comprehensive analysis that states, for every possible combination of NAT types, whether *direct* (i.e. with no relay) connectivity is possible and how. The analysis is more topical given that NAT community is switching from the classical set of NAT types $\mathcal{T}_{classic}$ = { Full-Cone, Restricted-Cone, Port-Restricted, Symmetric} [3] to a more elaborate set that defines a NAT type by a combination of three different policies, namely, port mapping, port allocation and port filtering [8]. With that, a statement like "two peers behind symmetric NAT can not communicate" becomes imprecise, as we will show that in many cases it is possible given the nuances available in the presently wide spectrum of NAT types.

II. Related Work

The work in [7] includes a matrix for a number of combinations, however mostly drawn from $\mathcal{T}_{classic}$ rather than the more elaborate classification in [8]. The work in [6] is probably the closest to ours, one can see our work as a superset of the set of combinations mentioned in that work.

III. NAT Types as Combinations of Policies

In this section we try to semi-formally summarize the more elaborate classification of NATs known as "BEHAVE-compliant"[8] and craft the notation that we will use in the rest of the paper.

Notation. Let n_a and n_b be NAT gateways. For $i \in \{a, b\}$, Let $P_i = \{p_i, p_i', p_i'', \dots\}$ be the set of peers behind n_i. An "endpoint" e is a host-port pair $e = (h, p)$, where $h(e)$ is the host of e and $p(e)$ is its port. Let $V_i = \{v_i, v_i', v_i'', \dots\}$ denote the set of all private endpoints of all peers behind n_i and $U_i = \{u_i, u_i', u_i'' \dots\}$ be the set of public endpoints of n_i. i.e $\forall v \in V_i, h(v) \in P_i$ and $\forall u : U_i, h(u) = n_i$.

When a packet is sent out from a certain private endpoint v_i of a peer p_i behind a gateway n_i, to some public endpoint d, a rule in the NAT table of n_i is created. We define the set of NAT table rules $R_i = \{r_i, r_i', r_i'''\}$ at n_i, the rule records the fact that some public port u_i and some private port v_i are associated, e.g $r_a = (v_a \leftrightarrow u_a)$.

The behavior of a gateway n_i is defined by three policies, namely, port mapping, port filtering and port allocation. We use the notation $f(n_i), m(n_i), a(n_i)$ to denote the respective policies of gateway n_i.

A. Mapping Policy

The mapping policy is triggered every time a packet is sent from a private endpoint v_i behind the NAT to some external public port d. The role of a mapping policy is deciding whether a new rule will be added or an existing one will be reused. We use the notation:

1) $\overrightarrow{v_i, d} \vDash r_i$ to specify that the sending of a packet from v_i to d resulted in the creation of a new NAT table rule r_i at n_i. That is the binding of a new public port on n_i. However, we say that a rule was created because we care not only about the binding of the port but also the constraints on using this new port.
2) $\overrightarrow{v_i, d} \Rightarrow r_i$ to specify that the sending of the packet reused an already existing rule r_i.
3) $\overrightarrow{v_i, d} \overset{reason}{\not\Rightarrow} r_i$ to specify that the *sending* of the packet did not reuse some r_i in particular because of some *"reason"*.

Irrespective of the mapping policy, whenever a packet is sent from a private port v_i to an arbitrary public destination endpoint d and $\nexists r_i \in R_i$ of the form $r_i = (v_i \leftrightarrow u_i)$, for an arbitrary u_i, the following is true $\overrightarrow{v_i, d} \vDash r_i$. However, if such a mapping exists, the mapping policy would make the reuse decision based on the destination. For all subsequent packets from v_i to d, naturally, $\overrightarrow{v_i, d} \Rightarrow r_i$. However, for any $d' \neq d$, there are 3 different behaviors:

- Endpoint-Independent, $m(n_i) =$ EI:
  $\overrightarrow{v_i, d'} \Rightarrow r_i$, for any d'
- Host-Dependent, $m(n_i) =$ HD:
  $\overrightarrow{v_i, d'} \Rightarrow r_i$, iff $h(d) = h(d')$
  $\overrightarrow{v_i, d'} \vDash r'_i$, iff $h(d) \neq h(d')$, where $r'_i = (v_i \leftrightarrow u'_i)$ and $u'_i \neq u_i$
- Port-Dependent, $m(n_i) =$ PD:
  $\overrightarrow{v_i, d'} \vDash r'_i$

Having introduced the different policies, we decorate the notation of the rule to include the criteria that will be used to decide whether a certain rule will be reused as follows:

$$r_i = \left\{ \begin{array}{ll} \left(v_i \xleftrightarrow[m:v_i \to *]{} u_i\right) & \text{if } m(n_i) = \text{EI} \\ \left(v_i \xleftrightarrow[m:v_i \to (h(d),*)]{} u_i\right) & \text{if } m(n_i) = \text{HD} \\ \left(v_i \xleftrightarrow[m:v_i \to d]{} u_i\right) & \text{if } m(n_i) = \text{PD} \end{array} \right\}$$

Where the syntax $m : x \to y$ means that the rule will be reused if the source endpoint of the packet is x and the destination is y. The $*$ denotes any endpoint.

Order. We impose the EI $<$ HD $<$ PD according to the increasing level of restrictiveness.

B. Allocation Policy.

Every time a new r_i is added to R_i, a new public endpoint u_i is bound. This policy allocates $p(u_i)$. That is, the mapping policy decides *when* to bind a new port and the allocation policy decides *which* port should be bound as follows:

1) Port-Preservation, $a(n_i) =$ PP:
   Given $\overrightarrow{v_i, d} \vDash r_i$, where $r_i = (v_i \leftrightarrow u_i)$, it is always the case that: $p(u_i) = p(v_i)$. Naturally, this may cause conflicts if any two p_i and p'_i behind n_i decided to bind private endpoints with a common port.
2) Port Contiguity, $a(n_i) =$ PC:
   Given any two sequentially allocated public endpoints u_i and u'_i it is always the case that: $p(u'_i) = p(u_i) + \Delta$, for some $\Delta = 1, 2, ...$
3) Random, $a(n_i) =$ RD:
   $\forall u_i, p(u_i)$ is allocated at random.

Order. We impose the order PP $<$ PC $<$ RD according to the increasing level of difficulty of handling.

C. Filtering Policy.

The filtering policy decides whether a packet from the outside world to a public endpoint of a NAT gateway should be forwarded to the corresponding private endpoint. Given an existing rule $r_i = (v_i \leftrightarrow u_i)$ that was created to send a packet from v_i to d, we use the notation:

1) $r_i \Leftarrow \overleftarrow{u_i, s}$ to denote that the *receival* of a packet from the public endpoint s to n_i's public endpoint u_i is permitted by r_i
2) $r_i \overset{reason}{\not\Leftarrow} \overleftarrow{u_i, s}$ to denote that the receival is not permitted because of some "reason".

There are 3 filtering policies with the following conditions for allowing receival:

- Endpoint-Independent, $f(n_i) =$ EI:
  $r_i \Leftarrow \overleftarrow{u_i, s}$, for any s
- Host-Dependent, $f(n_i) =$ HD:
  $r_i \Leftarrow \overleftarrow{u_i, s}$, iff $h(s) = h(d)$
- Port-Dependent, $f(n_i) =$ PD:
  $r_i \Leftarrow \overleftarrow{u_i, s}$, iff $s = d$

We also decorate the rules to include conditions for accepting packets as follows:

$$r_i = \left\{ \begin{array}{ll} \left(v_i \xleftrightarrow{f:u_i \leftarrow *} u_i\right) & \text{if } f(n_i) = \text{EI} \\ \left(v_i \xleftrightarrow{f:u_i \leftarrow (h(d),*)} u_i\right) & \text{if } f(n_i) = \text{HD} \\ \left(v_i \xleftrightarrow{f:u_i \leftarrow d} u_i\right) & \text{if } f(n_i) = \text{PD} \end{array} \right\}$$

Order. We impose the order EI $<$ HD $<$ PD according to the increasing level of restrictiveness.

D. The Set of NAT Types

Having defined the above policies, the NAT type of a given NAT gateway is simply a matter of listing which behavior is used for each of the policies. We define the set of triplets representing all possible NAT types $\tau = \{(m, a, f) | f, m \in \{\text{EI, HD, PD}\}, a \in \{\text{PP, PC, RD}\}\}$.

IV. NAT Type Discovery

Before traversing a NAT gateway, one needs to know its type. STUN [3] is the most-widely used method for accomplishing this and there exists many publicly-available

STUN servers that assist in the discovery process. The original STUN algorithm produces a classification withdrawn from the set $\tau_{classic}$. More recently, [6], [8] have re-used the STUN infrastructure to get more detailed information, namely, knowing the filtering and the mapping policies.

Due to space limitations and the fact that our main focus is on traversal strategies, we will not delve into the details of performing the discovery process. However, we just need to clarify that in the spirit of [6], [8], we have expanded the scope of the discovery process to discover information about the allocation policy. With that, our classification is capable of reporting all elements in the set τ.

V. NAT Traversal Techniques

We explain our traversal techniques which are an augmented version of the well-known techniques in [1].

Basic Assumptions. We assume that there is a Rendez-vous server with public IP referred to by z. The traversal process always starts after: $i)$ two Peers p_a and p_b respectively behind NATs n_a and n_b register themselves at z and have an "out-of-band" communication channel with z, which is in our case a TCP connection initiated by the peer, we refer to all endpoints of z and z itslef by the same symbol; $ii)$ The 2 peers know that they need to communicate and know the other peer's public IP, i.e. the corresponding NAT IP, some peers supply additional information during registration as we will shortly explain in Section VII-B; $iii)$ all the policies of p_a, p_b are known to z using a discovery process before any traversal process takes place.

VI. Simple hole-punching (SHP)

A. Traversal Process

1) p_a sends from some v_a to z through n_a.
2) n_a creates $r_a = (v_a \leftrightarrow u_a)$ and forwards to z.
3) z receives and consequently knows u_a.
4) z informs p_b about u_a (Out-of-band).
5) p_b sends from some v_b to u_a through n_b.
6) n_b creates $r_b = (v_b \leftrightarrow u_b)$ and forwards to u_a.
7) n_a receives, if the filtering allows, forwards to v_a
8) p_a sends from v_a to u_b through n_a, if the mapping allows, r_a is reused. Otherwise, r'_a will be created and sending will occur from some other public endpoint $u'_a \neq u_a$
9) n_b receives, if the filtering allows, forwards to v_b

B. SHP Feasibility

Theorem 6.1: Simple hole punching is feasible for establishing direct communication between two peers p_a and p_b respectively behind n_a and n_b if $\exists n_x \in \{n_a, n_b\}$ s.th. $f(n_x) =$ EI, and either $m(n_x) =$ EI or $m(n_x) >$ EI and $f(n_{x' \neq x}) <$ PD.

Proof: We consider the most restrictive case where $f(n_a) = f(n_b) = m(n_a) = m(n_b) =$ PD and $a(n_a) = a(n_b) =$ RD and show the minimum relaxations that we need to do for SHP to work. By looking at the steps in section VI, and considering all the very restrictive mapping and filtering on both sides, we can see that after steps 5 and 6, r_a and r_b will be as follows:

$r_a = \left(v_a \xleftarrow[m:v_a \rightarrow u_z]{f:u_a \leftarrow u_z} u_a\right), r_b = \left(v_b \xleftarrow[m:v_b \rightarrow u_a]{f:u_b \leftarrow u_a} u_b\right)$

Which will cause the following problems:

In step 7: $r_a \overset{u_b \neq u_z}{\not\Longleftarrow} \overleftarrow{u_b, u_a}$ and there is nothing that we can relax at n_b which can help. Instead, we have to relax the filtering at p_a to indulge receiving on u_a from u_b while it was initially opened for receiving from u_z. i.e, r_a has to tolerate host change which is not satisfied by PD nor HD filtering, therefore $f(n_a) =$ EI is necessary, resulting into $r_a = \left(v_a \xleftarrow[m:v_a \rightarrow u_z]{f:u_a \leftarrow *} u_a\right)$

In step 8: $\overrightarrow{v_a, u_b} \overset{u_b \neq u_z}{\not\Longrightarrow} r_a$ and $\overrightarrow{v_a, u_b} \vDash r'_a$ where $r'_a = \left(v_a \xleftarrow[m:v_a \rightarrow u_b]{f:u'_a \leftarrow *} u'_a\right)$. Consequently, $r_b \overset{u_a \neq u'_a}{\not\Longleftarrow} \overleftarrow{u'_a, u_b}$. To solve this, we have two solutions, the first is to let the mapping reuse r_a and not create r'_a which needs relaxing $m(n_a)$ to be EI, in which case we can keep $f(n_b)$ as restrictive. The second solution is to keep n_a as restrictive and relax $f(n_b)$ to tolerate receiving from u'_a. In the second solution, there is a minor subtlety that needs to he handled, where p_b has to be careful to keep sending to p_a on u_a despite the fact that it is receiving from u'_a. Similarly p_a should always send to p_b on u_b despite the fact it is receiving from u'_b. That is an asymmetry that is not in general needed.

■

C. Coverage of SHP

Since $|\tau| = 27$ types, we have a $\frac{27 \times 28}{2} = 378$ distinct combinations of NAT types of two peers. Using Theorem 6.1, we find that 186 combinations, i.e. 49.2% of the total number of possible ones are traversable using the Simple Hole Punching approach. That said, this high coverage is totally orthogonal to how often one is likely to encounter combinations in the covered set in practice, which we discuss in our evaluation (Section IX-A). Traversable SHP combinations are shown in Figure 1 with label SHP(*).

To cover the rest of the cases, we use port prediction which enables a peer to punch a hole by sending to the opposite peer instead of z, which makes it possible to tolerate more restrictive filtering and mapping policies, as explained below.

VII. Prediction

A. Prediction using Contiguity (PRC)

The traversal process consists in the following steps:

1) p_a sends two consecutive messages:
   - from some v_a to z through n_a
   - from v_a to u_b^{dum}, an arbitrary endpoint of n_b
2) n_a creates the following two rules:
   - $r'_a = (v_a \leftrightarrow u'_a)$ and forwards to z.
   - $r_a = (v_a \leftrightarrow u_a)$ and forwards to u_b^{dum}. Actually, the whole point of sending u_b^{dum} is to open u_a by sending to n_b but be able to predict it at z.
3) The messages are received as follows:

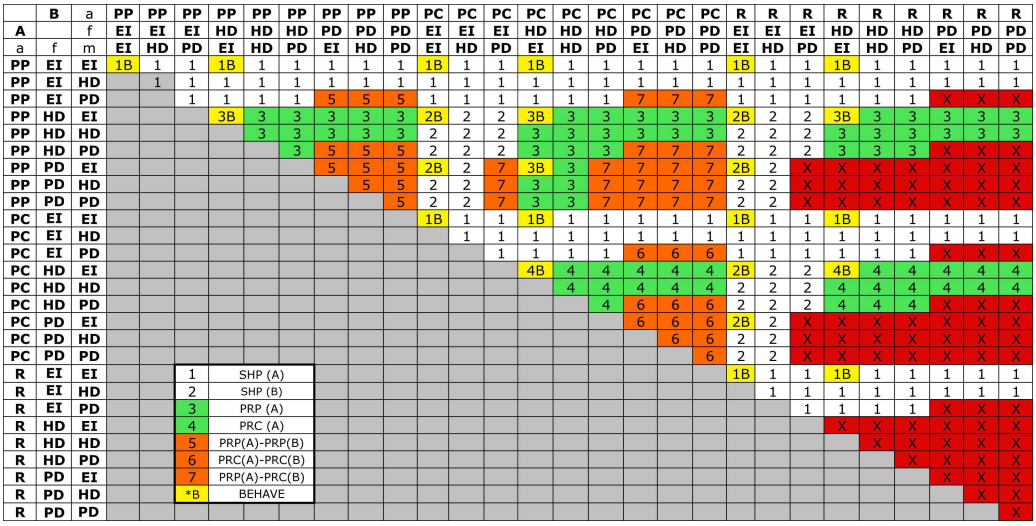

Fig. 1. All possible distinct NAT types combinations for two peers a and b with the technique needed to traverse the combination and X for un-traversable combinations. SHP(*), PRC(*) and PRP(*) stand respectively for Simple Hole Punching, Port Prediction using Contiguity and Port Prediction using Preservation. Combinations of NAT behaviors mandated by RFC 4787 are identified by the label BEHAVE in the table's legend.

a) z receives and consequently knows u'_a and additionally predicts $u_a = u'_a + \Delta$ where Δ is known during the discovery process.
b) n_b drops the message since no endpoint u_b^{dum} was ever bound.

4) z informs p_b about u_a (Out-of-Band).
5) Steps $5-9$ follow the same scheme as in simple hole punching.

Port scanning. The process is susceptible to failure if another peer p'_a happens by coincidence to send a packet between the two consecutive packets. For that, a technique called port scanning [6] is used such that when p_b tries to connect to u_a, p_b will try $u_a + \Delta, u_a + 2\Delta, u_a + 3\Delta$, etc.. until a reply is received. Some gateways might identify this as a malicious UDP port scan and block it as is the case in some corporate firewalls. Port scanning might be used only when p_b connecting to p_a where $a(n_a) = PC$ has $m(n_b) < PD$, as shown by[6].

B. Prediction using Preservation (PRP)

Another technique is to exploit the port-preservation allocation policy. However, to do that, we assume that when a peer with port-preservation policy registers at z, the peer supplies a pool of free candidate ports to z. The main point here is to avoid conflicts with ports of other peers behind the same NAT. The rendez-vous server z is stateful regarding which ports are bound by each NAT and chooses from the pool of the ports supplied by the peer a port which is not already bound.

1) z chooses some arbitrary port ρ and tells p_a (Out-of-Band) to bind ρ
2) p_a sends from v_a where $p(v_a) = \rho$ to u_b^{dum} through n_a.
3) n_a creates a new rule $r_a = (v_a \leftrightarrow u_a)u_b^{dum}$ and forwards to u_b^{dum} and since $a(p_a) = \text{PP}$, $p(u_a) = p(v_a) = \rho$.
4) z informs p_b about u_a (Out-of-Band).
5) Steps 5-9 follow the same scheme as in SHP.

Note that the process is shorter than prediction by contiguity and z chooses the port for the peer behind NAT instead of the NAT of the peer deciding it and z observing it. However, for the sake of reasoning, the two are equivalent because what matters is what happens after the opposite peer learns about the punched port irrespective of how the port was predicted.

C. Prediction-on-a-Single-Side Feasibility

Theorem 7.1: Prediction using contiguity or preservation on a single side is feasible for establishing direct communication between two peers p_a and p_b respectively behind n_a and n_b if:

- Condition 1: $\exists n_x \in \{n_a, n_b\}$ s.th. $a(n_x) < \text{RD}$ and $f(n_x) < \text{PD}$
- Condition 2: Either $m(n_x) < \text{PD}$ or $m(n_x) = \text{PD}$ and $f(n_{x' \neq x}) < \text{PD}$.

Proof: Similar to theorem 6.1, we start with the most restrictive policies and we relax until prediction is feasible. The allocation policy of the side to be predicted (n_a in Section VII-B,VII-A) can not be random, because the whole idea of prediction relies on a predictable allocation policy, thus the needed relaxation is $a(n_a) < \text{RD}$.

In both prediction techniques, the dummy packet from p_a punches a hole by sending to p_b, in contrast to SHP which punches by sending to z. nevertheless, it is sent to a dummy port of p_b. After steps 5, 6:

$$r_a = \left(v_a \xleftarrow[m:v_a \rightarrow u_p^{dum}]{f:u_a \leftarrow u_b^{dum}} u_a \right), r_b = \left(v_b \xleftarrow[m:v_b \rightarrow u_a]{f:u_b \leftarrow u_a} u_b \right)$$

In step 7: $r_a \overset{u_b \neq u_b^{dum}}{\not\Longleftarrow} \overleftarrow{u_b, u_a}$, we have to relax the filtering at p_a to indulge the port difference from u_b, but we tolerate host sensitivity. The needed relaxation is: $f(n_a) < \text{PD}$ resulting

into:

$$r_a = \left(v_a \xleftarrow[m:v_a \rightarrow u_b^{dum}]{f:u_a \leftarrow (n_b,*)} u_a \right)$$

In step 8: the reasoning about relaxing the mapping on p_a or the filtering of p_b is identical to Theorem 6.1 except that host-sensitivity is tolerable and thus either $m(n_a) <$ PD or is kept $m(n_a) =$ PD and in that case, the needed relaxation is $f(n_b) <$ PD. ■

D. Coverage of PRP & PRC

PRP and PRC together cover another 18% of the combinations. That said, we can say that PRP is as good as SHP in terms of traversal time and success rate (see Section IX), which means in addition to the cases where PRP on a single side is used in Figure 1, we can also use PRP instead of SHP when the allocation policy is port preservation.

VIII. Interleaved Prediction on Two Sides

. The remaining combinations are these not covered by SHP nor prediction. The final stretch to go is to do simultaneous prediction on both sides. However, it is a seemingly tricky deadlock situation because every peer needs to know the port that will be opened by the other peer without the other peer sending anything. Which we solve as follows.

Interleaved PRP-PRP. In this case actually double prediction is very simple because the rendez-vouz server can pick a port for each side and instruct the involved peers to simultaneously bind it and start the communication process.

Interleaved PRP-PRC This case is also easily solvable thanks to preservation. Because z can inform the peer with a port contiguity allocation policy about the specific endpoint of the opposite peer. The latter in turn will run a port prediction process using the obtained endpoint in the second consecutive message.

Interleaved PRC-PRC This one is the trickiest and it needs a small modification in the way prediction by contiguity is done. The idea is that the two consecutive packets, the first to z and the second to the opposite peer can not be sent after each other immediately. Instead, both peers are commanded by z to send a packet to z itself. From that, z deduces the ports that will be opened on each side in the future and sends to both peers informing them about the opposite peer's predicted endpoint. Both peers in their turn send a punching packet to each other. The problem with this scheme is that there is more time between the consecutive packets which makes it more susceptible to the possibility of another peer behind any of the NATs sending a packet in in between. Like the case in single PRC, port scanning is the only resort, but in general this combination has lower success rate compared to single PRC (see SectionIX).

For our reasoning, we will work on the last one (PRC-PRC), since it is a general harder case of the first two.

A. Traversal Process

1) z tells p_a & p_b to start prediction (Out-of-Band)

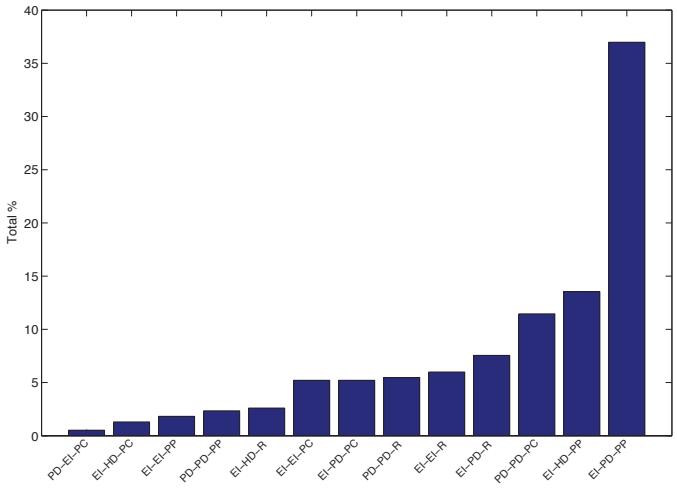

Fig. 2. Distribution of encountered NAT types in τ as (m,f,a)

2) p_a & p_b both send to z through n_a & n_b respectively resulting in the new rules $r'_a = (v_a \leftrightarrow u'_a), r'_b = (v_b \leftrightarrow u'_b)$
3) z receives from p_a & p_b, observing u'_a & u'_b and deducing $u_a = u'_a + \Delta$ & $u_b = u'_b + \Delta$
4) z informs p_a & p_b about u_b & u_a respectively (Out-of-Band)
5) p_a sends to u_b through n_a and p_b sends to u_a through n_b
6) n_b receives and forwards to v_b and n_a receives and forwards to v_a

A race condition where step 6 for one of the peers happens before the opposite peer starts to run step 5 can take place resulting into a packet drop. However, the dropped packet opens the hole for the opposite peer, and retrying sending is enough to take care of this issue.

B. Interleaved Prediction Feasibility

Theorem 8.1: Interleaved Prediction is feasible for establishing direct communication between two peers p_a and p_b respectively behind n_a and n_b if both $a(n_b)$ and $a(n_b)$ are $< RD$

Proof: Similar to theorem 6.1, we start with the most restrictive policies and we relax until prediction is feasible. Since we need to predict both sides we need $a(n_a) <$ RD & $a(n_b) <$ RD. After step 5 in Section VIII-A, we have:

$$r_a = \left(v_a \xleftarrow[m:v_a \rightarrow u_b]{f:u_a \leftarrow u_b} u_a \right), r_b = \left(v_b \xleftarrow[m:v_b \rightarrow u_a]{f:u_b \leftarrow u_a} u_b \right)$$

In step 6, we have $r_a \Leftarrow \overleftarrow{u_a, u_b}$ and $r_b \Leftarrow \overleftarrow{u_b, u_a}$ without the need for any relaxations on the filtering nor the mapping of either sides. ■

C. Interleaved Prediction Coverage

The interleaved prediction covers another 11.9% of the combinations, namely the ones shown in Figure 1 leaving 20.6% of the cases untraversable. That is, approximately 79.4% of all NAT type combinations are traversable and for

TABLE I
DISTRIBUTION OF ENCOUNTERED NAT POLICIES

Mapping	EI	HD	PD
	80.21%	0%	19.79%
Filtering	EI	HD	PD
	13.54%	17.45%	69.01%
Allocation	PP	PC	RD
	54.69%	23.7%	21.61%

each combination, we know which technique to use. The more important thing is that not all of them have the same likelihood of being encountered which we discuss in the next section. That said, it worth mentioning that there is a technique in [9] which performs a brute-force search on all possible ports after reducing the search space using the birthday paradox, which we ignored due to low success probability, high traffic and long time requirements.

IX. EVALUATION

Apart from the reasoning above, we have done a sanity check on our logic using our emulation platform [10]. That is, we wrote our own NAT boxes, which behave according to the semantics defined in Section III. We also implemented the Rendez-Vous server and the nodes that are capable of performing all the traversal techniques in Section V. For each case in Figure 1, we ran the suggested traversal technique and we made sure direct communication is indeed achievable. Real-life evaluation was needed to gain insights on other aspects like probability of encountering a given type, success rates of traversal techniques and time needed for the traversal process to complete.

A. Distribution of Types

We wanted to know how likely is it to encounter each of the types in τ. We have collected cumulative results for peers who have joined our network over time. As shown in Figure 2: i) we encountered 13 out of the 27 possible types; ii) we found that $(m = \text{EI}, f = \text{PD}, a = \text{PP})$ is a rather popular type (approx. 37%) of all encountered types, which is fortunate because port preservation is quite friendly to deal with and it is with a very relaxed mapping; iii) about 11% are the worst kind to encounter, because when two peers of this type need to talk, interleaved prediction is needed with a shaky success probability.

B. Adoption of BEHAVE RFC

By looking at each policy alone, we can see to what extent the recommendations of the BEHAVE RFC [8] ($f = \text{EI}/\text{HD}, m = \text{EI}$) are adopted. As shown in Table IX, for filtering, the majority are adopting the policy discouraged by the RFC, while for mapping the majority were following the recommendation. For allocation, the RFC did not make any specific relevant recommendation. The percentage of NATs following *both* recommendations was 30%.

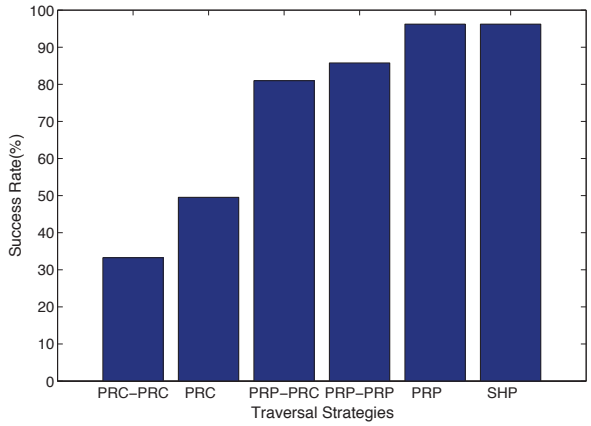

Fig. 3. Success rate of each technique averaged over all applicable combinations.

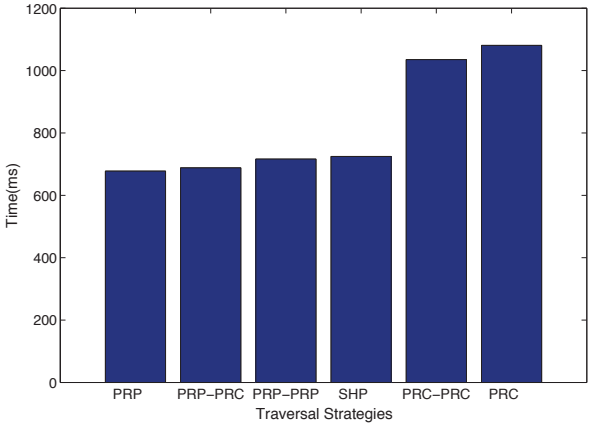

Fig. 4. Time taken (in msec) for the traversal process to complete.

C. Success Rate

Given the set of peers present in the network at one point in time, we conduct a connectivity test where all peers try to connect to each other. We group the result by traversal techniques, e.g. SHP is applicable for 186 combinations, so we average the success rate over all combinations and the whole process is repeated a number of times, we have found (Figure 3) as expected that SHP is rather dependable as it succeeds 96% of the time. We also found that PRP is as good as SHP, which is quite positive given that we found that the probability of occurrence of preservation is quite high in the last section. Interleaved PRP-PRP is also rather good with slightly worse success rate. The three remaining techniques involving PRC in a way or the other are causing the success rate to drop significantly especially for PRC-PRC mainly because of the additional delay for interleaving.

D. Time to traverse

When it comes to the time needed for the traversal process to complete (Figure 4), we find two main classes, SHP and

PRP in one class and PRC in another class, even when we do PRC-PRP, it is faster than PRC alone because the number of messages is less.

X. Conclusion & Future Work

In this paper, we have presented our experience with trying to find a comprehensive analysis of what combinations of NAT types are traversable. We have shown that using a semi-formal reasoning that covers all cases and we provided a slightly augmented versions of the well-known traversal techniques and shown which ones are applicable for which combinations.We have shown that about 80% of all possible combinations are traversable.

Using our deployment base for P2P live streaming, we have shown that only 50% fo all possible types are encounterable. We have also reported our findings on the success probability and time of traversing the different combinations.

For future work: a) Modeling: we would like to enrich the model to make it capture real-life aspects like expiration of NAT rules, multiple levels of NAT, subtleties of conflicts between many peers behind the same NAT, NATs that use different policies in different situations, and support for uPnP and TCP; b) Real-life Evaluation: more insight into the trade-off between success probability and timing, preventing the techniques as being identified as malicious actions in some corporate firewalls; c) Dissemination: releasing our library and simulator as open-source for third-party improvement and evaluation.

XI. Acknowledgments

We would like to thank all members of Peerialism's development team for the help and collaboration on the implementation of our NAT traversal techniques, in particular Magnus Hedbeck for his patience and valuable feedback. The anonymous reviewers of ICCCN have provided a really inspiring set of comments that helped us to improve the quality of this paper.

References

[1] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators," in *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2005, pp. 13–13.

[2] P. Srisuresh, B. Ford, and D. Kegel, "State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)," RFC 5128 (Informational), Internet Engineering Task Force, Mar. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5128.txt

[3] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," RFC 3489 (Proposed Standard), Internet Engineering Task Force, Mar. 2003, obsoleted by RFC 5389. [Online]. Available: http://www.ietf.org/rfc/rfc3489.txt

[4] C. H. J. Rosenberg, R. Mahy, "Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun)," Internet draft, November 2008. [Online]. Available: http://tools.ietf.org/html/draft-ietf-behave-turn-14

[5] J. Rosenberg, "Interactive connectivity establishment (ice): A protocol for network address translator (nat) traversal for offer/answer protocols," Internet draft, October 2007. [Online]. Available: http://tools.ietf.org/html/draft-ietf-mmusic-ice-19

[6] Y. Takeda, "Symmetric nat traversal using stun," Internet draft, June 2003. [Online]. Available: http://tools.ietf.org/html/draft-takeda-symmetric-nat-traversal-00

[7] D. Thaler, "Teredo extensions," Internet draft, March 2009. [Online]. Available: http://tools.ietf.org/html/draft-thaler-v6ops-teredo-extensions-03

[8] F. Audet and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," RFC 4787 (Best Current Practice), Internet Engineering Task Force, Jan. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4787.txt

[9] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig, "NATBLASTER: Establishing TCP connections between hosts behind NATs," in *Proceedings of ACM SIGCOMM ASIA Workshop*, Apr. 2005.

[10] R. Roverso, M. Al-Aggan, A. Naiem, A. Dahlstrom, S. El-Ansary, M. El-Beltagy, and S. Haridi, "Myp2pworld: Highly reproducible application-level emulation of p2p systems," in *Decentralized Self Management for Grid, P2P, User Communities workshop, SASO 2008*, 2008.

Appendix
Traversal Sequence Diagrams